Understanding Constraint Expressions in Large Conceptual Schemas by Automatic Filtering

Antonio Villegas, Antoni Olivé, and Maria-Ribera Sancho

Department of Service and Information System Engineering
Universitat Politècnica de Catalunya – BarcelonaTech
Barcelona, Spain
{avillegas, olive, ribera}@essi.upc.edu

Abstract. Human understanding of constraint expressions (also called schema rules) in large conceptual schemas is very difficult. This is due to the fact that the elements (entity types, attributes, relationship types) involved in an expression are defined in different places in the schema, which may be very distant from each other and embedded in an intricate web of irrelevant elements. The problem is insignificant when the conceptual schema is small, but very significant when it is large. In this paper we describe a novel method that, given a set of constraint expressions and a large conceptual schema, automatically filters the conceptual schema, obtaining a smaller one that contains the elements of interest for the understanding of the expressions. We also show the application of the method to the important case of understanding the specification of event types, whose constraint expressions consists of a set of pre and postconditions. We have evaluated the method by means of its application to a set of large conceptual schemas. The results show that the method is effective and efficient. We deal with conceptual schemas in UML/OCL, but the method can be adapted to other languages.

Keywords: Constraints, Large Conceptual Schemas, Filtering

1 Introduction

The conceptual schema of many real-world information systems is large or very large. General ontologies (like Cyc [1]), metaschemas (like UML superstructure [2]) or information reference models (like HL7 [3]) are also very large. In general, those conceptual schemas or ontologies include many formal constraint expressions (also called schema rules), which are used for defining static or dynamic integrity constraints, derivation rules, default values, pre and postconditions of events and operations, or results of operations.

In large conceptual schemas, human understanding of such expressions is difficult. The problem is not the formal language in which they are written (logic in general, or the OCL in UML [4]), but the fact that the elements (entity types, attributes, relationship types) involved in an expression are defined in different places in the schema, which may be very distant from each other and embedded

in an intricate web of irrelevant elements for the purpose at hand. The problem is insignificant when the schema is small, but very significant when it is large.

As a very simple example, consider a schema with the n-level specialization hierarchy B_n IsA ... B_i IsA ... B, the binary relationship type *R(b:B,c:C)* and the attribute *att(C,Integer)*. Assume that in the context of B_n there is the simple constraint expression `self.c.att > 0`. Understanding such expression requires finding entity type B_n (the context) in the schema, moving upwards the n-level hierarchy until the root (entity type B), navigating towards C, and finding attribute *att*. When n is large and each of the entity types B_n, ... B_i, ... B, C has several attributes and participates in several relationship types, the task of localizing the relevant elements in the constraint expression, and focusing on them, becomes difficult.

The overall framework of the research is that of design science [5], where knowledge and understanding of a problem domain and its solution are achieved in the building and application of a designed artifact. The problem we try to solve is to ease the understanding of constraint expressions in large conceptual schemas. The expressions are written in a constraint language, such as the OCL. The problem is significant because:

- in a conceptual schema there may be many constraint expressions, used as –among others– invariants, derivation rules, and pre/post conditions,
- understanding such expressions is a necessity during their definition, validation, implementation, and maintenance, and
- understanding such expressions is very difficult when the conceptual schema is large, because the elements involved in those expressions are in general scattered throughout the schema, and it is not easy to navigate through the schema in the way implied by the expressions.

In this paper we describe a method that, given a set of constraint expressions and a large conceptual schema, automatically filters the conceptual schema, obtaining a smaller one that contains the elements of interest for the understanding of the expressions. We have evaluated our method by means of its application to a set of large conceptual schemas, consisting of the UML metaschema [2, 6], the Magento [7] and osCommerce [8] schemas from the e-commerce domain, and the schema of the car rental case study known as EU-Rent [9].

As far as we know, this is the first work that filters conceptual schemas with the objective of easing the understanding of constraint expressions. The method is based and extends the work reported in [10], whose aim was to filter a large schema focusing on one or more entity types and to enrich them with other schema elements according to their importance. Here, we focus on one or more constraint expressions and we obtain a schema that contains all elements that appear in them, independently of their importance. In the simple example introduced above, our method would obtain a filtered schema consisting of entity types B_n and C, the binary relationship type *R(b:B_n,c:C)* and the attribute *att(C,Integer)*. Note that the hierarchy B_n IsA ... B_i IsA ... B does not appear in the filtered schema, and that the first participant of the relationship type R has changed to B_n.

In the literature, there are several techniques and associated tools for the visualization and comprehension of large conceptual schemas or ontologies (see [11–18]). The group of techniques more appropriate for our purposes is the one called focus+context. In these techniques, the user focuses on a single element, and the rest of the elements are presented around it, reduced in size until they reach a point that they are no longer visible. In our approach, we filter the schema with the objective of obtaining a subset of the original one that contains all relevant elements.

The rest of the paper is structured as follows. The next section introduces the basic concepts and notations of the schemas considered in this paper. Section 3 describes our filtering method. The evaluation of the method in terms of effectiveness and efficiency is presented in Sect. 4. Section 5 explains the adaptation of the method to the special case of filtering event types. Finally, Sect. 6 concludes the paper and points out to future work.

2 Conceptual Schemas

In the general case, a conceptual schema $\mathcal{CS}$ contains the following components:

- A a set of entity types $\mathcal{E}$.
- A set of event types $\mathcal{B}$.
- A set of data types and enumeration types $\mathcal{T}$.
- A set of relationship types $\mathcal{R}$. We denote by $R(p_1{:}C_1, ..., p_n{:}C_n)$ a relationship type R with participant entity or event types $C_1, ..., C_n \in \mathcal{E} \cup \mathcal{B}$ playing roles $p_1, ..., p_n$ respectively.
- A set of attributes $\mathcal{A}$. We see attributes as binary relationship types. We denote by *attr(C,T)* an attribute owned by an entity or event type $C \in \mathcal{E} \cup \mathcal{B}$, named *attr*, and whose type is $T \in \mathcal{T}$.
- A set of generalization relationships $\mathcal{G}$ between entity or event types. We denote by $g{:}(C_i \text{ IsA } C_{i-1}) \in \mathcal{G}$ the generalization relationship g between C_i and C_{i-1}. $\mathcal{G}^+$ is the transitive closure of generalization relationships.
- A set of schema rules $\mathcal{S}$ including integrity constraints, derivation rules, default values, pre and postconditions of events and operations, and results of operations.

In the rest of the paper, we assume that schemas are written in UML/OCL, although the filtering method presented here could be used with schemas and formal constraint expressions written in other languages.

We illustrate the method by means of its application to the Magento[1] e-commerce system [7]. Its conceptual schema (see a snapshot in Fig. 1) contains 218 entity types and 187 event types, with 983 attributes, 319 relationship types, and 165 generalization relationships. Also, Magento's schema defines 61 data types and enumeration types. The set of schema rules includes 480 integrity constraints, 69 pre and postconditions, and 185 derivation rules.

[1] Magento e-commerce System `http://www.magentocommerce.com/`

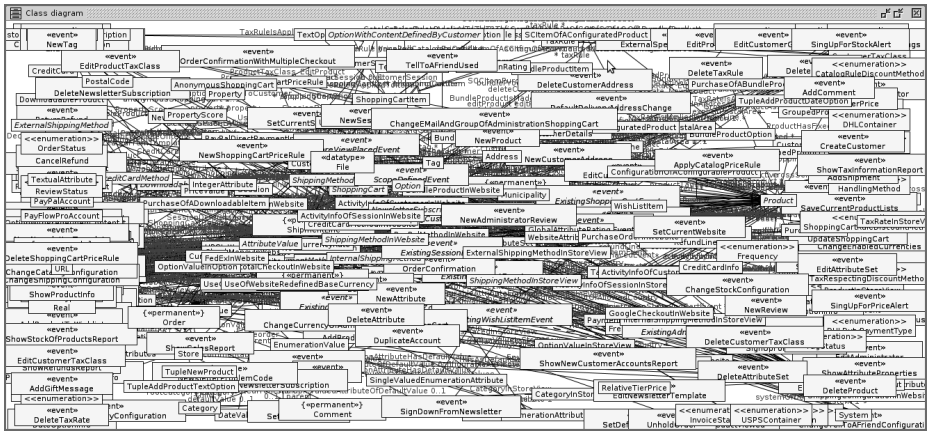


Fig. 1. Graphical representation of the conceptual schema of the Magento system.

3 Filtering Conceptual Schemas

The aim of information filtering is to expose users to only information that is relevant to them. There are many filtering systems of widely varying philosophies [18], but all share the goal of automatically directing the most valuable information to users in accordance with their needs, and of helping them use their limited time and information processing capacity most optimally.

In [10] we proposed a method to deal with large conceptual schemas, in which a user focuses on one or more entity types of interest for her task, and the method filters the schema in order to obtain a reduced and self-contained view from it, consisting of the focus and a few schema elements that may be of interest for her based on the automatically computed importance of those elements.

In the method we propose here, the user focuses on a set of schema rules, which may be integrity constraints, derivation rules, or pre/post conditions, and the method obtains the smallest subset of the schema that is needed to understand those expressions. Figure 2 presents an overview of our filtering method.

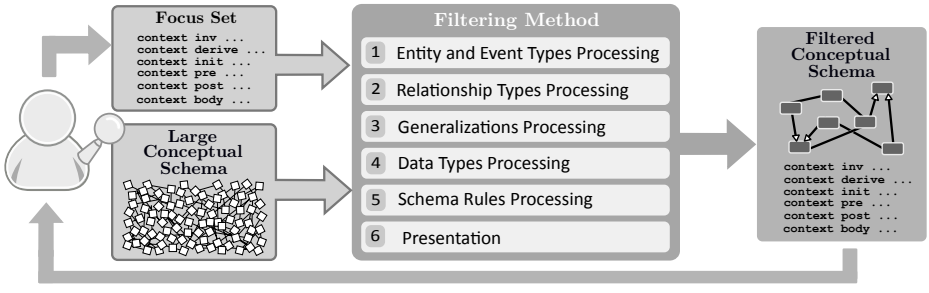


Fig. 2. Method Overview.

3.1 Input of the Filtering Method

The components of the input for the filtering method are:

- **Large Conceptual Schema ($\mathcal{CS}$):** It represents the knowledge about a domain of interest. It has the components presented in Sect. 2.
- **Focus Set ($\mathcal{FS}$):** It works as the conceptual schema viewpoint of the user. Formally, the focus set $\mathcal{FS}$ contains one or more constraint expressions selected by the user.

3.2 Output of the Filtering Method

The output of our filtering method is a filtered conceptual schema $\mathcal{CS}_{\mathcal{F}}$=$\langle \mathcal{E}_{\mathcal{F}}$, $\mathcal{B}_{\mathcal{F}}$, $\mathcal{T}_{\mathcal{F}}$, $\mathcal{R}_{\mathcal{F}}$, $\mathcal{A}_{\mathcal{F}}$, $\mathcal{G}_{\mathcal{F}}$, $\mathcal{S}_{\mathcal{F}}\rangle$. The contents of such conceptual schema depends on the particular user information needs represented in the input of the filtering method. The properties $\mathcal{CS}_{\mathcal{F}}$ must satisfy are:

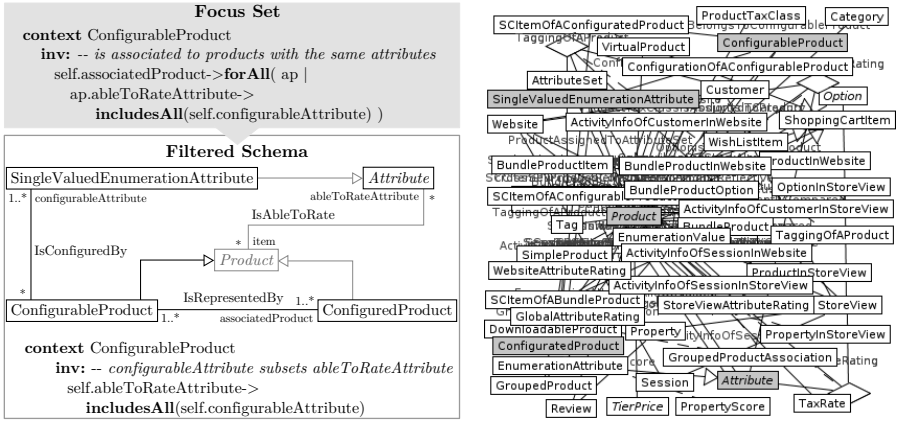
- **Minimal Subset:** The knowledge contained in $\mathcal{CS}_{\mathcal{F}}$ is the smallest subset of the knowledge of $\mathcal{CS}$ referenced by the constraint expressions of $\mathcal{FS}$. The filtering method does not create new knowledge through the filtering process. The schema elements that appear in the filtered schema come from the original large schema through a process of knowledge extraction.
- **Valid Instantiation:** The filtered conceptual schema is a valid instance of the corresponding metaschema of the original schema from which it is obtained. Thus, the filtered schema is syntactically correct since the elements included within it are concrete instances of metaclasses of that metaschema.

3.3 Stages of the Filtering Method

The filtering method is divided into six ordered stages that sequentially process the input specified by a user in order to obtain the corresponding filtered conceptual schema. As an example, we assume that the user needs to understand the following integrity constraint defined in the Magento, in the context of the entity type *ConfigurableProduct*:

```
context ConfigurableProduct
   inv: -- is associated to products with the same attributes
     self.associatedProduct->forAll(ap |
       ap.ableToRateAttribute->includesAll(self.configurableAttribute))
```

Figure 3(a) presents the filtered schema our method will obtain for the previous integrity constraint. A *ConfigurableProduct* allows customers to configure some of its attributes when purchasing it (such as color, size...). It is related to a set of *ConfiguredProducts*, each one representing one concrete available configuration for the *ConfigurableProduct*. The constraint indicates that the set of attributes that are able to be rated in a *ConfiguredProduct* must contain all the configurable attributes of its *ConfigurableProducts*. The additional constraint included in the filtered schema only references elements in $\mathcal{CS}_{\mathcal{F}}$ and indicates that the configurable attributes of a *ConfigurableProduct* must be included in its set of attributes that are able to be rated.



(a) Filtered conceptual schema ($\mathcal{CS}_{\mathcal{F}}$). (b) Fragment of Magento.

Fig. 3. Comparison between (a) the filtered conceptual schema and (b) the corresponding fragment of Magento for the constraint of *ConfigurableProduct*.

The size of the filtered schema is smaller than the fragment of the Magento concerning the elements of entity types referenced by the schema rule of focus, which is depicted in Fig. 3(b). Note that it also contains 193 attributes (not shown in Fig. 3(b)), of which 33 are owned by entity types in $\mathcal{CS}_{\mathcal{F}}$. By using our method, a user does not need to manually explore the schema in Fig. 3(b) in order to extract the required elements to understand the schema rules of focus.

Next subsections present a detailed description of the stages of the filtering method, and their application to obtain the filtered schema shown in Fig. 3(a).

Stage 1: Filtering of Entity and Event Types. The method firstly extracts the entity and event types referenced by the selected schema rules of $\mathcal{FS}$ and includes them in the resulting filtered conceptual schema. Formally,

$$\mathcal{E}' = \{e \in \mathcal{E} \mid \exists s \ (s \in \mathcal{FS} \wedge s \rightsquigarrow e)\},$$
$$\mathcal{B}' = \{b \in \mathcal{B} \mid \exists s \ (s \in \mathcal{FS} \wedge s \rightsquigarrow b)\}$$

We say that $s \rightsquigarrow x$ whenever $s \in \mathcal{FS}$ and x is (a) the context entity or event type of s, (b) an entity or event type being the target participant of a relationship type in a navigation expression of s, or (c) an attribute, entity, event, enumeration, or data type explicitly referenced in s.

Considering the integrity constraint from the running example (see Fig. 3.3), $\mathcal{E}'$ consists of the entity types *ConfigurableProduct* (context of the constraint), *ConfiguredProduct* (target participant of `self.associatedProduct` navigation), *Attribute* (target participant of `ap.ableToRateAttribute` navigation), and *SingleValuedEnumerationAttribute* (target participant of `self.configurableAttribute` navigation).

Stage 2: Filtering of Relationship Types and Attributes. The method extracts the relationship types and the attributes referenced by the selected schema rules of $\mathcal{FS}$ and includes them in the resulting filtered conceptual schema. For the case of relationships, we define the set of referenced relationship as, $\mathcal{R}' = \{R(p_1{:}C_1, ..., p_n{:}C_n) \in \mathcal{R} \mid \exists s, p_i{:}C_i \ (s \in \mathcal{FS} \wedge s \rightsquigarrow p_i{:}C_i \wedge p_i{:}C_i \in R)\}$.

Considering the running example, the method extracts the following three referenced relationship types for $\mathcal{R}'$:

```
IsRepresentedBy(configurableProduct:ConfigurableProduct,
                associatedProduct:ConfiguredProduct),
IsConfiguredBy(configurableProduct:ConfigurableProduct,
               configurableAttribute:SingleValuedEnumerationAttribute),
IsAbleToRate(item:Item,ableToRateAttribute:Attribute).
```

The next step consists in projecting the participants of the referenced relationships $\mathcal{R}'$ to the entity or event types of the filtered schema. Formally, given a participant $p{:}C$ we define,

$$projection(p{:}C) = \begin{cases} p{:}C & \text{if} \quad C \in \{\mathcal{E}' \cup \mathcal{B}'\} \\ p{:}LCA(\mathcal{D}_C) & \text{if} \quad C \notin \{\mathcal{E}' \cup \mathcal{B}'\} \end{cases},$$

$$\text{where} \ \ \mathcal{D}_C = \{C_i \in \{\mathcal{E}' \cup \mathcal{B}'\} \mid C_i \ \text{IsA}^+ \ C\}.$$

The lowest common ascendant (LCA) of a set of siblings $C_1, ..., C_n \in \{\mathcal{E}' \cup \mathcal{B}'\}$ that descend from the participant C is the lowest entity or event type in the hierarchy shared by the siblings that has all $C_1, ..., C_n$ as descendants (where we allow an element to be a descendant of itself). Whenever the set of descendants $\mathcal{D}_C$ is empty, the corresponding LCA equals to C. If the LCA obtained by projecting is not a member of $\mathcal{E}' \cup \mathcal{B}'$, the method includes it as an auxiliary entity or event type in $\mathcal{E}_\mathcal{X}$ or $\mathcal{B}_\mathcal{X}$. Therefore, the final entity and event types of the filtered schema are $\mathcal{E}_\mathcal{F} = \mathcal{E}' \cup \mathcal{E}_\mathcal{X}$ and $\mathcal{B}_\mathcal{F} = \mathcal{B}' \cup \mathcal{B}_\mathcal{X}$. For the case of relationships, the final set $\mathcal{R}_\mathcal{F}$ contains the projections of the relationship types in $\mathcal{R}'$. Formally, $\mathcal{R}_\mathcal{F} = \{R(projection(p_1{:}C_1), ..., projection(p_n{:}C_n)) \mid R \in \mathcal{R}'\}$. The method performs the same filtering and projection steps for the attributes explicitly referenced by schema rules of $\mathcal{FS}$.

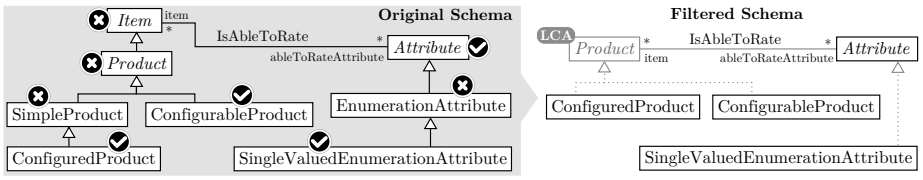


Fig. 4. Projection of the referenced relationship type *IsAbleToRate*.

Figure 4 presents the referenced relationship type *IsAbleToRate* defined between *Item* and *Attribute*, and its projection to the LCA of the descendants of *Item* in the filtered schema. Note that since *Product* is the LCA of both *ConfigurableProduct* and *ConfiguredProduct*, and it was not filtered into $\mathcal{E}'$ in the previous stage, the method includes it in the auxiliary set $\mathcal{E}_\mathcal{X} \subset \mathcal{E}_\mathcal{F}$.

Considering the integrity constraint from the running example, the set of filtered attributes $\mathcal{A}_\mathcal{F}$ is empty since the constraint does not reference any attribute. The final filtered relationship types from the projection of the relationships of $\mathcal{R}'$ in $\mathcal{R}_\mathcal{F}$ are:

```
IsRepresentedBy(configurableProduct:ConfigurableProduct,
                associatedProduct:ConfiguredProduct),
IsConfiguredBy(configurableProduct:ConfigurableProduct,
               configurableAttribute:SingleValuedEnumerationAttribute),
IsAbleToRate(item:Product,ableToRateAttribute:Attribute).
```

Stage 3: Filtering of Generalization Relationships. At this point of the filtering process, the resulting filtered schema contains its entity, event, relationship types, and attributes. Now, we determine the needed generalization relationships between the filtered entity and event types. The method automatically selects the direct generalizations between filtered entity or event types, and includes them into the filtered schema. However, it is possible to have a pair of filtered entity or event types e_i, e_j so that e_i is an indirect ascendant of e_j in the original schema but they are not connected through generalization relationships in the filtered one. To avoid this inconsistency, the method creates auxiliary generalizations between those unconnected pairs. Formally, $\mathcal{G}_\mathcal{F} = \{g{:}(C_i \text{ IsA } C_j) \mid C_i, C_j \in \{\mathcal{E}_\mathcal{F} \cup \mathcal{B}_\mathcal{F}\} \wedge \exists g'(g'{:}(C_i \text{ IsA}^+ C_j) \in \mathcal{G}^+)\}$.

Figure 5 shows the hierarchy of *Product* referenced by the integrity constraint of the running example. Since this constraint does not reference the entity type *SimpleProduct*, it is not included into the resulting filtered schema. First, the method obtains the direct generalization relationship between *ConfigurableProduct* and *Product*. As a result, the entity type *ConfiguredProduct* that was a descendant in that hierarchy of the original schema, it is now an isolated entity type in the filtered schema. Next, in order to avoid that situation and maintain the original semantics, our method creates an auxiliary generalization in order to show that *ConfiguredProduct* is indirectly a descendant of *Product*. Similarly, the method includes an auxiliary generalization between *SingleValuedEnumerationAttribute* and *Attribute* in $\mathcal{G}_\mathcal{F}$.

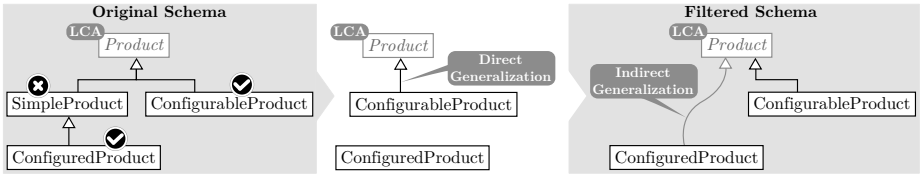


Fig. 5. Filtering of the generalization relationships from the hierarchy of *Product*.

Stage 4: Filtering of Data Types. The method includes in the filtered schema those data types from the original one that are referenced in the specification of a schema rule of $\mathcal{FS}$ or define the type of a filtered attribute of $\mathcal{A}_\mathcal{F}$. Formally, $\mathcal{T}_\mathcal{F} = \{T \in \mathcal{T} \mid \exists s\ (s \in \mathcal{FS} \wedge s \rightsquigarrow T) \vee \exists a, C\ (C \in \{\mathcal{E}_\mathcal{F} \cup \mathcal{B}_\mathcal{F}\} \wedge\ a(C, T) \in \mathcal{A}_\mathcal{F}\}$.

In the running example, the set $\mathcal{T}_\mathcal{F}$ is empty since the constraint does not reference any data type.

Stage 5: Filtering of Schema Rules. The method selects the schema rules defined in the context of elements from $\mathcal{CS}_{\mathcal{F}}$ and only includes in $\mathcal{S}_{\mathcal{F}}$ those rules that are referentially-complete. A schema rule is referentially-complete whenever all the schema elements used in the specification of the rule belong to the filtered conceptual schema. All the schema rules from the focus set are referentially-complete since the schema elements they reference are all already in the filtered schema. Formally, $\forall s\ ((s \in \mathcal{S} \wedge \neg\exists x(s \rightsquigarrow x \wedge x \notin \mathcal{CS}_{\mathcal{F}})) \Rightarrow s \in \mathcal{S}_{\mathcal{F}})$. The execution of this stage is performed only if the user wants in $\mathcal{S}_{\mathcal{F}}$ the additional schema rules.

Considering the running example, the method includes in $\mathcal{S}_{\mathcal{F}}$ the integrity constraint of $\mathcal{FS}$, and an additional constraint from $\mathcal{S}$ that only references the filtered elements in the former stages of the method (see Fig. 3(a) bottom).

Stage 6: Presentation of the Filtered Schema. The last step of the filtering method graphically presents to the user the elements included in the resulting filtered conceptual schema, as shown in Fig. 3(a). Note that the projected relationship types and the auxiliary entity and event types from Stage 2, and the auxiliary generalizations from Stage 3, are marked with a lighter color.

3.4 Method Correctness

It can be shown that $\mathcal{CS}_{\mathcal{F}}$ is minimal and a valid instance of the metaschema. The reason is that it only includes those elements from $\mathcal{CS}$ that truly participate in the specification of the schema rules of focus, apart from the additional schema rules from Stage 5 that are included only on user-demand. By projecting relationship types and attributes, and avoiding unnecessary generalization relationships, our method creates a filtered schema with standard constructions that cannot be smaller for a given input focus set. Any deletion of an element in the filtered schema produces an inconsistency in such schema with respect to the specification of the schema rules of focus.

4 Evaluation

Finding a measure that reflects the ability of our method to satisfy the user is a complicated task in the field of information retrieval [19]. Usually, the distinction is made between the evaluation of the effectiveness and the efficiency of a retrieval method. While the effectiveness measures the benefits obtained from the application of the filtering method, the efficiency indicates the time interval between the request being made and the answer being given.

We have implemented the filtering method described in the previous section as an extension of the prototype tool described in [20]. We have then evaluated the efficiency and effectiveness of the method by using four distinct case studies: the schema of Magento [7], the UML metaschema [2, 6], the schema of osCommerce [8], and the EU-Rent car rental schema [9]. Table 1 summarizes the components of these four schemas. We have applied our filtering method for each schema rule specified in these conceptual schemas. In the following, we present the results we have obtained from the analysis of the resulting data.

Table 1. Components of the conceptual schemas used in the evaluation.

$\mathcal{CS}$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{B}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{G}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{S}\rvert$
Magento	218	187	983	319	165	61	734
UML metaschema	293	0	93	377	355	13	170
osCommerce	84	262	209	183	393	17	457
EU-Rent	65	120	85	152	207	7	283

4.1 Effectiveness Analysis

Our method produces a filtered conceptual schema of small size that helps understanding the schema rules of a large schema. We compare the final size of the filtered schema with the size of the context constraint schema, i.e, the portion of the large schema the user needs to manually explore in order to cover the elements referenced by the formal specification of the schema rules of focus.

For a set of schema rules of focus $\mathcal{FS}$, we define the components of the constraint context schema $\mathcal{CS}_\mathcal{C} = \langle \mathcal{E}_\mathcal{C}, \mathcal{B}_\mathcal{C}, \mathcal{R}_\mathcal{C}, \mathcal{A}_\mathcal{C}, \mathcal{G}_\mathcal{C} \rangle$ as:

$$\mathcal{E}_\mathcal{C} = \mathcal{E}_\mathcal{S} \cup \mathcal{E}_\mathcal{G} \cup \mathcal{E}_\mathcal{R},$$
$$\mathcal{B}_\mathcal{C} = \mathcal{B}_\mathcal{S} \cup \mathcal{B}_\mathcal{G} \cup \mathcal{B}_\mathcal{R},$$
$$\mathcal{R}_\mathcal{C} = \{R(p_1{:}C_1, ..., p_n{:}C_n) \in \mathcal{R} \mid \exists C_i, p_i \ (C_i \in \{\mathcal{E}_\mathcal{C} \cup \mathcal{B}_\mathcal{C}\} \wedge p_i{:}C_i \in R)\},$$
$$\mathcal{A}_\mathcal{C} = \{a \in \mathcal{A} \mid \exists C, T \ (C \in \{\mathcal{E}_\mathcal{C} \cup \mathcal{B}_\mathcal{C}\} \wedge T \in \mathcal{T} \wedge a(C, T))\},$$
$$\mathcal{G}_\mathcal{C} = \{g \in \mathcal{G} \mid \exists C_i, C_j \ (C_i, C_j \in \{\mathcal{E}_\mathcal{C} \cup \mathcal{B}_\mathcal{C}\} \wedge g{:}(C_i \text{ IsA } C_j))\},$$

where $\mathcal{E}_\mathcal{S}$ and $\mathcal{B}_\mathcal{S}$ are the entity and event types referenced by the schema rules of focus. For the example of Fig. 4, $\mathcal{E}_\mathcal{S} = \{$*ConfigurableProduct*, *Configured-Product*, *Attribute*, *SingleValuedEnumerationAttribute*$\}$. $\mathcal{E}_\mathcal{G}$ and $\mathcal{B}_\mathcal{G}$ contain the entity and event types that are intermediate members of the paths of generalization relationships between members of $\mathcal{E}_\mathcal{S}$ and $\mathcal{B}_\mathcal{S}$. For the example of Fig. 5, $\mathcal{E}_\mathcal{G} = \{$*SimpleProduct*$\}$. $\mathcal{E}_\mathcal{R}$ and $\mathcal{B}_\mathcal{R}$ are the participant entity and event types in relationship types and attributes referenced by schema rules from the focus set, without applying projection. For the example of Fig. 4, $\mathcal{E}_\mathcal{R} = \{$*Item*, *Attribute*$\}$.

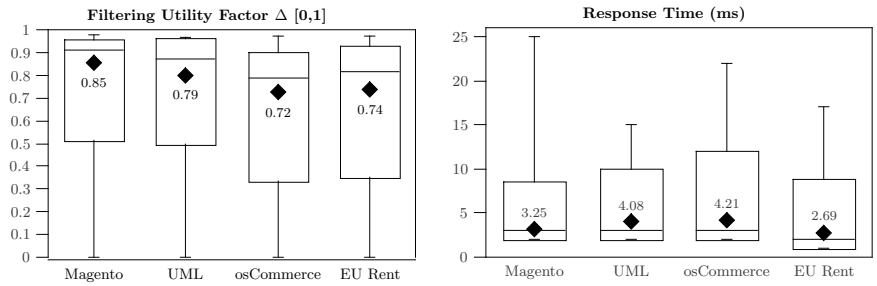
Therefore, we define the filtering utility factor between the filtered schema $\mathcal{CS}_\mathcal{F}$ and the context constraint schema $\mathcal{CS}_\mathcal{C}$ as follows:

$$\text{Filtering Utility Factor: } \Delta = 1 - \frac{\Sigma(\mathcal{CS}_\mathcal{F})}{\Sigma(\mathcal{CS}_\mathcal{C})}, \text{ where}$$
$$\Sigma(\mathcal{CS}_\mathcal{F}) = |\mathcal{E}_\mathcal{F}| + |\mathcal{B}_\mathcal{F}| + |\mathcal{R}_\mathcal{F}| + |\mathcal{A}_\mathcal{F}| + |\mathcal{G}_\mathcal{F}|, \text{ and}$$
$$\Sigma(\mathcal{CS}_\mathcal{C}) = |\mathcal{E}_\mathcal{C}| + |\mathcal{B}_\mathcal{C}| + |\mathcal{R}_\mathcal{C}| + |\mathcal{A}_\mathcal{C}| + |\mathcal{G}_\mathcal{C}|.$$

Figure 6(a) presents a box plot with the resulting values for the filtering utility factor applied to each of the 1644 schema rules of the case studies. For each schema, the plot indicates the smallest observation (sample minimum), lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation (sample maximum). Also, the black diamonds indicate the mean of each sample.

The bottom and top of each box (Q1 and Q3) are the 5th and 95th percentiles, which means that the box contains the 90% of the samples.

We observe that the mean value of the filtering utility factor exceeds 0.7 in any case, which indicates a size reduction greater than 70% using filtered schemas instead of working manually. It implies a significant reduction of the cognitive effort a user has to face when understanding the schema rules of a large schema.



(a) Filtering utility factor. (b) Response Time.

Fig. 6. Effectiveness and efficiency analysis.

The smallest observations of the filtering utility factor in any schema ($\Delta = 0$) indicate that the size of the filtered schema equals the size of the context constraint schema ($\Sigma(\mathcal{CS}_{\mathcal{F}}) = \Sigma(\mathcal{CS}_{\mathcal{C}})$). This situation occurs whenever the schema rule of focus only references all the attributes of a single entity or event type without relationships to other elements, as in the case of primary key constraints of isolated types. In our experimentation, the schema rules that cause this represent less than a 2% of the total schema rules analyzed by the method.

4.2 Efficiency Analysis

It is clear that a good method does not only need to be useful, but it also needs to obtain the results in an acceptable time according to the user's expectations. To find the time spent by our method it is only necessary to record the time lapse between the request of knowledge, i.e. once a focus set $\mathcal{FS}$ containing schema rules has been indicated by the user, and the obtainment of the filtered schema.

Figure 6(b) presents a box plot with the resulting values for the response time (in milliseconds) obtained by an Intel Core 2 Duo 3GHz processor with 4GB of DDR2 RAM. The mean value of the response time is less than 5 milliseconds, which indicates that the time a user expends waiting for the resulting filtered schema is negligible. Furthermore, the largest observations of the response time in any schema are below 40 milliseconds. It is expected that as the number of projections of relationship types and subsumed generalization relationships to process increases, the response time will increase linearly. However, the resulting times for all the schema rules of the case studies are short enough for our purpose.

5 Application to Event Types

Our method can be used to help understanding the specification of event types in large conceptual schemas. We see this as an important application of the method. An event describes a nonempty set of changes in the population of entity or relationship types in the domain of the conceptual schema. We assume event types are represented as entity types with the stereotype «event». This allows one to define relationships between events and other entities, integrity constraints, derivation rules, etc. in a way very similar to that for ordinary entity types [21].

We define an *effect()* operation in each event type, whose purpose is to specify the effect of the event in the domain. The postcondition of this operation will be exactly the postcondition of the corresponding event. We can use a constraint language like OCL to specify these postconditions. Then, we can directly apply our filtering method to the a set of postconditions of a particular event type in order to obtain its corresponding filtered schema.

Figure 7 (left) presents the two postconditions –*addProduct* and *decrease-Quantity*– of the *AddProductToShoppingCart* event, and the resulting filtered schema (right) that our method automatically obtains ($\Delta = 0.8$ in 3ms). Such schema can be seen as an Effect Correspondence Diagram (ECD) [22] that shows all the entities affected by a given event type.

6 Conclusions

We have focused on the problem of understanding constraint expressions in large conceptual schemas, in which the elements referenced by the expressions may be very distant from each other and embedded in an intricate web of irrelevant elements for this purpose.

We have proposed a filtering method in which a user focuses on a set of schema rules and the method obtains a filtered schema that includes the smallest subset of the original schema that is needed to understand those expressions. We have implemented our method in a prototype tool and we have evaluated it by means of its application to four large conceptual schemas. The results show that our method achieves a size reduction greater than 70% in the number of schema elements to explore when understanding a schema rule by using filtered schemas instead of working manually, with an average time per request that is short enough for the purpose at hand. We have also shown the application of the method to help understanding the specification of event types.

We plan to extend our method in order to be capable of processing a filtered conceptual schema as the input focus set. The method will construct an enriched schema with the knowledge of the filtered schema and a few schema elements with relation to the schema rules from which the filtered schema was obtained. The resulting schema may be of interest for a user that has to modify a schema rule and needs to explore the fragment of the large schema that involves the elements affected by the schema rule and a set of additional elements to which they are related.

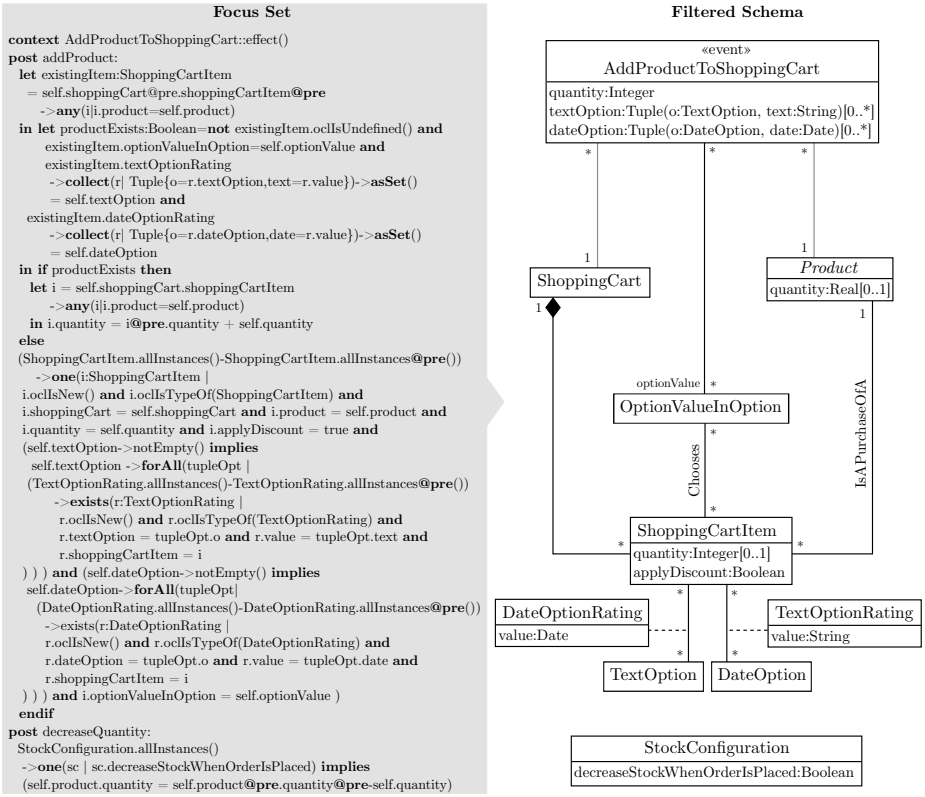


Fig. 7. Application of the filtering method to the event *AddProductToShoppingCart*.

Acknowledgements

This work has been partly supported by the Ministerio de Ciencia y Tecnologia and FEDER under project TIN2008-00444/TIN, Grupo Consolidado, and by Universitat Politècnica de Catalunya under FPI-UPC program.

References

1. Lenat, D.B.: Cyc: a large-scale investment in knowledge infrastructure. Commun. ACM **38**(11) (1995) 33–38
2. Object Management Group (OMG): Unified Modeling Language (UML) Superstructure Specification, version 2.3. (May 2010)
3. Beeler, G.: HL7 Version 3–An object-oriented methodology for collaborative standards development. International Journal of Medical Informatics **48**(1-3) (1998) 151–161
4. Object Management Group (OMG): Object Constraint Language Specification (OCL), version 2.0. (February 2010)
5. Hevner, A., March, S., Park, J., Ram, S.: Design science in information systems research. Mis Quarterly **28**(1) (2004) 75–105

6. Bauerdick, H., Gogolla, M., Gutsche, F.: Detecting OCL traps in the UML 2.0 Superstructure: An experience report. In: UML 2004 – The Unified Modeling Language. Modelling Languages and Applications. Volume 3273 of Lecture Notes in Computer Science. Springer (2004) 188–196
7. Ramirez, A.: Esquema conceptual de Magento, un sistema de comerç electrònic. Technical report, Universitat Politècnica de Catalunya, `http://hdl.handle.net/2099.1/12294` (2011)
8. Tort, A., Olivé, A.: The osCommerce conceptual schema. Technical report, Universitat Politècnica de Catalunya, `http://hdl.handle.net/2099.1/5301` (2007)
9. Frias, L., Queralt, A., Olivé, A.: EU-Rent car rentals specification. Technical report, Universitat Politècnica de Catalunya, `http://www.lsi.upc.edu/~techreps/files/R03-59.zip` (2003)
10. Villegas, A., Olivé, A.: A method for filtering large conceptual schemas. In: Conceptual Modeling – ER 2010. Volume 6412 of Lecture Notes in Computer Science., Springer (2010) 247–260
11. Tzitzikas, Y., Hainaut, J.L.: How to tame a very large ER diagram (using link analysis and force-directed drawing algorithms). In: Conceptual Modeling – ER 2005. Volume 3716 of Lecture Notes in Computer Science. Springer (2005) 144–159
12. Auddino, A., Dennebouy, Y., Dupont, Y., Fontana, E., Spaccapietra, S., Tari, Z.: SUPER – Visual interaction with an object-based ER model. In: Entity-Relationship Approach – ER'92. Volume 645 of Lecture Notes in Computer Science. Springer (1992) 340–356
13. Lanzenberger, M., Sampson, J., Rester, M.: Visualization in ontology tools. In: Intl. Conf. on Complex, Intelligent and Software Intensive Systems, IEEE Computer Society (2009) 705–711
14. Shoval, P., Danoch, R., Balabam, M.: Hierarchical entity-relationship diagrams: the model, method of creation and experimental evaluation. Requirements Engineering **9**(4) (2004) 217–228
15. Moody, D.L., Flitman, A.: A methodology for clustering entity relationship models – a human information processing approach. In: Conceptual Modeling – ER'99. Volume 1728 of Lecture Notes in Computer Science. Springer (1999) 114–130
16. Campbell, L.J., Halpin, T.A., Proper, H.A.: Conceptual schemas with abstractions making flat conceptual schemas more comprehensible. Data & Knowledge Engineering **20**(1) (1996) 39–85
17. Kuflik, T., Boger, Z., Shoval, P.: Filtering search results using an optimal set of terms identified by an artificial neural network. Information Processing & Management **42**(2) (2006) 469–483
18. Hanani, U., Shapira, B., Shoval, P.: Information filtering: Overview of issues, research and systems. User Modeling and User-Adapted Interaction **11**(3) (2001) 203–259
19. Belkin, N.J., Croft, W.B.: Information filtering and information retrieval: two sides of the same coin? Commun. ACM **35** (December 1992) 29–38
20. Villegas, A., Sancho, M.R., Olivé, A.: A tool for filtering large conceptual schemas. In: Advances in Conceptual Modeling – ER 2011 Workshops. Volume 6999 of Lecture Notes in Computer Science., Springer (2011) 353–356
21. Olivé, A.: Definition of events and their effects in object-oriented conceptual modeling languages. In: Conceptual Modeling – ER 2004. Volume 3288 of Lecture Notes in Computer Science. Springer (2004) 136–149
22. Downs, E., Clare, P., Coe, I.: Structured Systems Analysis and Design Method: Application and Context. Second edn. Prentice Hall (1992)